In the Specification

Please replace paragraph **[0008]** with new paragraph **[0008]**, shown below.

[0008] One approach is to have a user initiate code generation for the interfaces only when the interface is actually needed. In this case, a client application may run a code generation tool to statically generate a ~~java~~ JAVA™ file and compile the ~~java~~ JAVA™ file into a class file. The user then makes the class available to the ~~Java~~ JAVA™ virtual machine. This approach is undesirable because it requires a user to manually perform many steps to generate the code.

Please replace paragraph **[0009]** with new paragraph **[0009]**, shown below.

[0009] Another method for generating interfaces involves ~~Java~~ JAVA™ dynamic proxies. ~~Java~~ JAVA™ dynamic proxies require that an interface and an invocation handler be provided by a user. In return, the ~~Java~~ JAVA™ dynamic proxy generation system provides a class that will forward invocations to the invocation handler. ~~Java~~ JAVA™ dynamic proxies are limited in the types of classes that they generate. It is not possible, for example, to generate a class that is a subclass of a user defined class. Another problem with dynamic proxies is that they are not as efficient as dynamic code generation. To implement many types of proxies, it is often necessary to use reflection within the invocation handler. Reflection is not as efficient as early bound invocation. There is also a cost associated with the way that dynamic proxies marshal the arguments for an invocation into an object array.

Please replace paragraph **[0014]** with new paragraph **[0014]**, shown below.

[0014] In one embodiment, the ~~Java~~ JAVA™ based automatic program code generator may be used to generate code for any type of ~~Java~~ JAVA™ program. The invention is especially useful

when used to build efficient adapters and proxies. Applications of the ~~Java~~ JAVA™ automatic code generator include but are not limited to remote method invocation (RMI) skeletons, RMI stubs, wrappers for JDBC connections, and proxies used to enforce call-by-value semantics between EJBs, the latter of which are applied to copying parameters. Typically, the code implementing a proxy or adaptor is dynamically generated when the code is needed, such as when a remote method is invoked on a resource. However, the dynamic code generation of the present invention may occur at any time depending on the particular application and resource available.

Please replace paragraph **[0016]** with new paragraph **[0016]**, shown below.

[0016] A method is then added to the class file object at step 120. At generation, the method is empty and contains no code. Step 120 may be repeated several times depending on the number of methods that will be contained within the class file. For example, for a stub generated for a remote object, the stub may include several methods. In this case, for each method in the remote interface, a method would be added to the new class file container object. Code may then be added to the method at step 130. In one embodiment, code is added to a method using constructs that correspond to ~~Java~~ JAVA™ language statements, expressions, variables, or any other programming elements. Each of these constructs may include parameters as necessary.

Please replace paragraph **[0019]** with new paragraph **[0019]**, shown below.

[0019] After the class file container object methods and code have been added, ~~Java~~ JAVA™ bytecode may be generated at step 140. In operation, each statement maintains the state of the program being generated. The maintained state includes, among other things, the contents of the stack and the contents of the local variables that are in use at each point of the program flow. The

statement uses this state to generate an intermediate representation of the program flow that consists of ~~Java~~ JAVA™ objects that represent individual bytecode instructions. A bytecode assembler converts the intermediate representation into bytecode that can be interpreted by a ~~Java~~ JAVA™ virtual machine.

Please replace paragraph **[0024]** with new paragraph **[0024]**, shown below.

[0024] The class above is a simplified example that is configured to generate text that reads, "Hello World!" The pseudo code above illustrates code for generating two types of expressions, an invoke expression and a return statement. This illustration is for illustrative purposes only. Other expressions, statements, variables, and other programming constructs are within the scope of the present invention. These programming constructs may include, but are not limited to, switch statements, array expressions (such as an expression allowing one to index into an array), cast expressions (allowing a cast from one object type to another), compound statements (a list of statements), conditional expressions (including Boolean expressions), constant expressions (for all primitive programming types), invoke expressions (for invoking methods on different types of objects), expressions that represent local variables of a method, and expressions for creating new objects and arrays. In one embodiment, the code generation tool of the present invention may be configured to include an expression that represents each ~~Java~~ JAVA™ programming expression and a statement that represents each ~~Java~~ JAVA™ programming statement.